

18

QUE LE FORTH SOIT AVEC VOUS

NOVEMBRE 1985

MUMPS
LPB
Langage C
KERMIT

pour

AMSTRAD
IBM
VAX

KERMIT

langage C

EDITORIAL

Ce mois-ci JEDI lance un pavé dans la mare des programmes, un pavé de taille puisque nous diffusons le programme KERMIT, écrit en langage C. Cette diffusion est faite en collaboration avec l'association O.U.F. (Ordinateurs Utilisateurs France) dont Mr Bill GRAHAM est le président.

KERMIT est disponible sur la base de de données de O.U.F. au standard V21 (300 bds) en composant le (1) 45.31.57.25.

Pourquoi KERMIT et qu'est-ce qu'il représente ? Il faut rappeler que lors d'une liaison télématique, des erreurs de transmission peuvent survenir. Si celles-ci sont sans gravité pour une information sous forme de texte, il n'en est pas de même pour un programme en code objet destiné à être exécuté. Ainsi, il est nécessaire de faire appel à des procédures de vérifications et de corrections. Jusqu'à présent, c'était le programme XMODEM qui réalisait cette tâche pour les systèmes 8 bits bits sous CP/M.

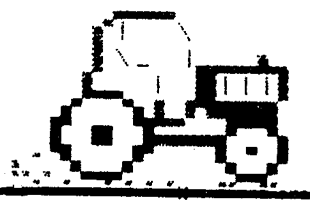
KERMIT permet la liaison au même protocole que XMODEM, mais appliqué aux systèmes 16 bits de type IBM et autres sous UNIX. Grâce à un programme de référence rigoureux écrit en langage C, il s'applique à une très large gamme de matériel (lourd, tant pis pour les petits systèmes) allant des systèmes IBM à VAX, permettant une transmission sûre des fichiers depuis les systèmes 8 vers 16 bits et 16 bits entre eux.

A part ce programme fleuve, nous continuons notre série sur MUMPS et sur LPB. Pour les amateurs de FORTH, ne soyez pas déçus, reportez-vous au mensuel MICRO-SYSTEMES de ce mois, nous avons aussi dense que KERMIT, mais écrit en FORTH: Composeur VIDEOTEX pour TO7 et TO7/70. A signaler que ce programme est en soi un petit record du monde, car, à notre connaissance, il est le premier programme FORTH de cette taille à être diffusé dans la presse informatique en dehors des articles interclubs américains.

Et pour finir, un grand merci à tous ceux qui nous ont envoyé leurs articles pour les numéros à venir: un article sur PILOT et des trucs pour HRX et JUPITER ACE en décembre; la virgule flottante en 83-Standard et une analyse détaillée de la vectorisation pour tous standards en janvier; fBASE I, un programme de gestion de fichier en FORTH pour tous systèmes. Ceci pour vous mettre l'eau à la bouche. Et nous laisserons de la place pour les articles sur PASCAL, MUMPS, COBOL, LISP (si, si!!), et...FUTURLOG (non ???). Etonnant n'est-ce pas ...

SOMMAIRE

MUMPS:	5ème partie	2
LPB:	7ème leçon	7
Langage C: KERMIT		9



ASSOCIATION JEDI 8, rue Poirier de Narçay 75014 PARIS
Tel: (1) 45.42.88.90 (de 10h à 18h)

VII INTRODUCTION AUX COMMANDES

A) Qu'est ce qu'une commande ?

Considérons les trois phrases suivantes :

arrêtez !
ouvrez la porte !
mettez un livre sur la table !

Chaque phrase est un ordre qui indique à quelqu'un de faire quelque chose. En parlant à certaines personnes nous adoucissons ces ordres en disant : "S'il vous plait, ouvrez la porte !". Mais quand nous nous exprimons de la sorte, l'ordre devient plus ambigu. "S'il vous plait, arrêtez !" peut être tout à fait différent de l'ordre "arrêtez !", selon les circonstances.

Lorsqu'on communique avec un ordinateur, nous devons lui donner des ordres aussi clairs que possible (soit, des ordres directs et peut être impolis) pour nous faire comprendre. Par chance, la machine préfère cette forme d'expression. Les premieres phrases citées sont sans ambiguité, si nous savons qu'il n'y a qu'une seule porte, un seul livre et une seule table. La premiere phrase est un long mot qui indique simplement l'ordre de faire quelque chose. Dans le second cas, on demande à quelqu'un de faire quelque chose en se servant d'autre chose. Enfin, dans la troisième phrase, nous demandons à quelqu'un d'accomplir quelque chose en se servant d'autre chose. Il serait possible d'inventer des ordres beaucoup plus complexes mais, pour le moment, ces trois exemples suffisent.

Le langage MUMPS permet de passer des ordres identiques à ceux de notre langage courant. Certaines commandes se composent d'un seul mot (ex. la commande halt). D'autres commandes demandent à l'ordinateur d'exécuter une action sur un ou plusieurs éléments. Ces derniers seront les arguments de la commande (verbe et complément(s)). Si la commande est suivie de plusieurs arguments, ceux-ci seront séparés par une virgule.

B) La commande SET

Cet ordre indique à MUMPS qu'il doit attribuer une valeur à une variable (contenu/contenant). Exemples :

```
SET X=3              le contenu de X est égal à 3
SET A="2 VELOS",B="3 MOTOS"
```

Dans le dernier exemple, un seul verbe a été mentionné, mais deux actions ont été effectuées. Soit, l'attribution à la variable A du contenu "2 VELOS" et à B le contenu "3 MOTOS". Il est à noter que si les variables A et B n'existaient pas, elles sont automatiquement créées. Si une de ces variables existait deja, c'est son contenu qui sera changé. Si, par exemple, X avait déja été défini avec un contenu égal à "ABC", après l'exécution de l'ordre SET X=3, son contenu devient 3. Bien entendu, on peut également affecter à une variable le résultat d'un calcul. Exemple : SET Y=X+7, le contenu de Y est égal a 10. N'oubliez pas que MUMPS ne fait pas de difference entre les valeurs alphanumériques et les valeurs numériques. Si X n'avait pas été défini au préalable, MUMPS enverrait un message d'erreur. D'autre part, les opérateurs de concaténation peuvent également etre utilisés. Exemple :

```
SET NOM="PERSONNE",TITRE="MON NOM EST "_NOM
```

Une autre utilisation du verbe SET permet, grâce à sa formulation, de donner à plusieurs variables, le même contenu. Exemple :

```
SET (X,Y,Z)=20        ; chacune des variables X, Y, Z contient la
                        valeur 20
```

Le dernier cas d'utilisation de SET peut être le suivant :

```
SET X=5,Y=2,Z=X*Y,W=X>Y
```

Dans cet exemple, X aura pour valeur 5, Y sera égal à 2 et Z le résultat de leur multiplication, c'est à dire 10. Et W aura la valeur vraie, à savoir 1, puisque la dernière expression X>Y est juste. Il faut remarquer, que chaque argument est évalué de gauche à droite.

C) La commande SET $PIECE

La révision de 1982 permet d'utiliser la commande SET en conjonction avec la fonction $PIECE pour définir ou redéfinir le segment d'une variable. Par exemple :

```
SET MOIS="30/12/1695",$PIECE(MOIS,"/",2)="01"
```

Le nouveau contenu de la variable MOIS est maintenant "30/01/1695".
Si vous aviez des problèmes de compréhension, reportez-vous au paragraphe traitant la fonction $PIECE.

D) La commande KILL

Cet ordre permet de supprimer une variable préalablement définie par la commande SET. On peut l'utiliser de différentes façons : Avec un seul argument, KILL supprime la variable mentionnée ; suivie de plusieurs arguments séparés par des virgules, KILL supprime les variables citées ; sans argument, elle enlève toutes les variables définies au préalable.
Exemples :

ordre d'exécution	effet produit
KILL A	variable A supprimée
KILL B,C	B et C supprimées
KILL	toutes les variables definies sont supprimées

Une autre formulation de l'ordre KILL est la suivante : KILL (A,B,C) ; dans ce cas, toutes les variables autres que A,B,C sont supprimées. Donc, A B et C sont conservées.

Attention ! KILL (A),(B) supprime toutes les variables puisque cette formulation équivaut à : KILL (A) ; donc la variable B est enlevée et KILL (B) enlève toutes les variables sauf B mais, B n'existe déja plus.

E) La fonction $SELECT

Cette fonction spéciale retourne la valeur de la première expression située la plus à gauche dans sa liste d'arguments dont l'expression équivalente est verifiée. Chaque argument de la fonction $SELECT est une paire d'expressions separées par deux points (:). La partie gauche de la paire est une variable logique et la partie droite peut etre n'importe quelle expression. Pour illustrer ceci, considérons la situation suivante :

```
A=1 ; B=2 ; C=3

SET RESULTAT=$SELECT(A>B:1,A=B:2,A*B:3)
```

$SELECT analyse la premiere expression de chaque argument, de gauche à droite. Lorsqu'elle trouve une expression vraie c'est à ce moment, et seulement à cet instant précis, que la seconde expression est évaluée comme résultat. Puisque, dans notre exemple A est plus petit que B ; qu'il n'est pas égal à la variable B ; la troisième expression, qui a pour valeur 3, est prise en compte et la valeur 3 est affectée à la variable RESULTAT.

La fonction $SELECT peut être également utilisée en dernier ressort, dans le cas où tous les autres critères sont faux, alors la valeur par défaut est retournée. Gardons les mêmes variables et examinons l'exemple ci-dessous :

```
SET RESULTAT=$SELECT(A=B:4,A=C:3,1:0)
```

Lorsque la machine ne pourra pas exécuter un ordre correctement, un message d'erreur explicite sera envoyé à l'utilisateur. Le contenu des messages peuvent varier d'une machine à l'autre.

Maintenant, examinons quelques exemples supplémentaires d'utilisation de la fonction $SELECT avec les variables précitées :

code d'execution	valeur produite
SET RESULTAT=$SELECT(A>B:1,A>C:2,A*B:3)	3
SET RESULTAT=$SELECT(A=B:1,B=C:2,A=C:3)	erreur
SET RESULTAT=$SELECT(A=B:2*C,A=C:A+B,1:0)	0
SET RESULTAT=$SELECT(A>B:A,A*B:C,A*C:B)	3 (valeur de C)
SET RESULTAT=$SELECT(A=C:"LB1",A*C:"LB2",1:"LB3")	"LB2"

Dans le dernier exemple deux cas sont vrais, mais $SELECT choisit le premier cas vrai qu'il rencontre, l'évaluation étant réalisée de la gauche vers la droite.

F) la commande WRITE (avec contrôle de format possible)

Nous avons déja examiné une commande associée à une fonction. Maintenant, voyons une commande "type" et certaines de ses options. La commande WRITE est l'une des deux commandes principales d'entrée/sortie. Elle permet de visualiser une chaîne de caractères sur la console de l'utilisateur. Exemple

```
WRITE "GEORGES WASHINGTON TRAVERSAIT LE POTOMAC"
```

Cette commande reproduit la même phrase sur l'écran de l'utilisateur.

Si nous voulions nous servir de MUMPS comme d'une calculette, il suffirait d'ecrire, par exemple :

```
WRITE 2*3+4," ",20+(20*12/100)
```

Le resultat affiché à l'ecran sera le suivant :

```
10 22.4
```

Bien entendu, nous pouvons utiliser des variables dans la commande WRITE. Si X a été defini auparavant (X=3), on peut imaginer l'exemple suivant :

```
WRITE "la valeur de X est : ",X
```

Le résultat affiché à l'écran sera le suivant :

```
la valeur de X est : 3
```

Dans ce cas, puisque A n'est pas égal aux variables B et C, la valeur attribuée à la variable RESULTAT est 0 (option par defaut). La fonction doit toujours avoir un argument vrai, sinon une erreur sera émise.

Afin de permettre à l'utilisateur de placer un message sur l'écran ou sur une feuille de listing, des codes de formattage sont disponibles.

Le code de contrôle ! apparaissant dans une commande WRITE, impliquera que le message suivant ce code sera imprimé au debut de la ligne suivante. Par conséquent, la commande suivante :

```
WRITE "hello",!,"vous êtes sous le contrôle de mumps"
```

Produit, sur l'écran, le résultat suivant :

```
hello
vous êtes sous le contrôle de mumps
```

→

Il est également possible d'utiliser un autre caractère spécial qui est le diese (#). Ce caractère a pour action, pour un écran, de placer le curseur en haut à gauche de celui-ci (1ere ligne/colonne 1), pour une imprimante, de changer de page.

Un autre caractère spécial est disponible dans la commande WRITE ; il s'agit du point d'interrogation (?), suivi d'un nombre entier, qui a pour effet de positionner le message à une colonne definie par le nombre entier.
Exemple :

```
WRITE #,?20,"bonjour"
```

L'effet produit sera : le positionnement du curseur à la ligne 1, colonne 1 de l'écran, puis, le positionnement du curseur en colonne 20 et l'écriture du message "bonjour" en colonne 21. On peut également trouver l'ordre suivant, mais nous vous laissons deviner l'action produite :

```
WRITE #!!,"CLIENT",?10,NOM
```

Nous espérons que vous avez trouvé...

Néanmoins, nous allons vous donner la solution :

Le curseur se positionnera en haut à gauche de l'écran, descendra de deux lignes, enverra le message "CLIENT", se positionnera à la colonne 10, puis écrira le contenu de la variable NOM en colonne 11.

G) Les variables spéciales $X et $Y

Ces variables sont des variables système alimemtées par la commande write. Le contenu de $X sera toujours incrémenté du nombre de caractères émis depuis le dernier envoi du signe !, celui-ci remettant à zéro la variable $X. On peut dire, pour $X, qu'il s'agit de la variable compteur de colonnes.

Au contraire, la variable $Y est la variable de compteur lignes. Elle est remise a zéro lors de l'émission du signe #. La variable $X est également remise a zéro.

H) La commande READ

Nous avons presenté la commande WRITE avant la commande READ parce que, en effet, la commande READ permet à l'utilisateur d'écrire un message sur le terminal avant de demander d'entrer des donnees au clavier. Cette possibilite est particulièrement utile pour prévenir l'utilisateur que l'ordinateur est entrain d'attendre des informations. La commande READ est construite pratiquement de la meme manière que la commande WRITE, à la différence près que l'on cite la ou les variables réceptrices des données entrées au clavier. L'exemple suivant illustre une commande READ avec des arguments multiples :

```
READ !,"Age : ",AG,!,"date de naissance (jj/mm/aa) : ",DDN
```

L'exécution de cette commande se déroulera de la manière suivante :

- passage du curseur à la ligne suivante
- envoi du message Age :
- attente de saisie au clavier
- la variable AG sera alimentée par la saisie
- passage du curseur à la ligne suivante
- envoi du message date de naissance (jj/mm/aa) :
- attente de saisie au clavier
- alimentation de la variable DDN par la saisie

Bien entendu, les caractères de controle #, !, ? peuvent être employés dans une commande READ.

La nouvelle norme de 1982 a étendu les fonctionnalités de la commande READ. Nous les étudierons ultérieurement.

I) Postconditions sur les commandes

MUMPS permet au programmeur de postconditionner pratiquement toutes les commandes en utilisant une expression qui sera évaluée en tant que valeur vraie. Si elle est trouvée juste, la commande est exécutée.

Prenons, par exemple, la commande : SET X=X+1 chaque que fois que la commande est exécutée, elle ajoute 1 à la valeur courante de X. Supposez que vous ne vouliez pas que la valeur de X dépasse 10, il existe un moyen d'accomplir cet objectif, c'est d'ajouter une postcondition à la commande d'origine. Nous ecrirons la commande de la manière suivante :

```
SET:X<11 X=X+1
```

En clair cette forme d'écriture correspond à SET X=X+1 si X est inférieur à 11.

La postcondition est énoncée derrière le verbe en les séparant par deux points (:). Cette facilité n'est quasiment propre qu'à MUMPS.

Les postconditions ne peuvent pas s'appliquer aux trois commandes MUMPS (IF, ELSE et FOR) que nous décrirons plus tard.

Si l'on veut avoir une information à l'écran, lors d'un traitement itératif, et que le message mentionnant le nombre de données traitées ne soit envoyé que tous les cent passages, on écrirait l'ordre suivant :

```
WRITE:N#100=0 !,N," donnees traitées"
```

J) Possibilité d'abréviation des commandes

MUMPS permet de citer les commandes ou les fonctions de deux façons différentes. La première est d'écrire la commande ou la fonction en entier, tel que : SET, KILL, WRITE, $LENGTH, $SELECT, $FIND etc... la deuxième consiste à écrire seulement la première lettre de la commande ou les deux premiers caractères de la fonction. Les commandes et les fonctions que nous avons exposées auparavant peuvent cependant etre abregèes comme suit :

KILL	K
READ	R
SET	S
WRITE	W
$ASCII	$A
$CHAR	$C
$EXTRACT	$E
$FIND	$F
$LENGTH	$L
$PIECE	$P
$RANDOM	$R
$SELECT	$S

Voici quelques exemples d'abréviation de commandes :

```
R "veuillez entrer la date du jour : ",DATE
W A,!,C,#,D,$L(Z),$P(I,Q,3)
R X,R,Z
```

Dans la derniere commande on pourrait penser qu'il y a un risque de confusion entre le verbe READ (R) et la variable R. Mais MUMPS solutionne ce genre de confusion dans sa sémantique, car les verbes seront séparés des arguments par un ou plusieurs espaces et les arguments multiples séparés par des virgules.

Lorsqu'une commande n'a besoin d'aucun argument, celle-ci devra impérativement être suivie de deux espaces. Sauf dans le cas particulier, ou une telle commande se trouve seule dans une ligne ou à la fin d'une ligne.

Christian SCHERER

Des cercles sur votre écran

Nous avons vu lors des leçons précédentes comment créer certaines primitives graphiques réellement performantes en langage-machine pour :

```
PLOT x,y        : allumer le pixel de coordonnées (x,y)
LINE x,y        : tirer un trait d'extrémité (x,y)
```

Aujourd'hui, nous emprunterons à notre ami mathématicien Jean LEFLOUR une solution élégante et inédite permettant de tracer un cercle à une vitesse inégalée.

L'algorithme employé évite le recours à la trigonométrie, et même au calcul de racine carrée, bien qu'il se réfère à la formule de Pythagore. Tous les calculs s'effectuent en nombres entiers.

On se ramène d'abord à la construction d'un huitième de périmètre, les sept autres s'en déduisant au fur et à mesure par des symétries évidentes.

Le premier point de l'arc aura pour coordonnées cartésiennes :

x=r   y=0

On décrémente ensuite progressivement x. Pour chaque décrémentation de x, on incrémente y jusqu'à ce que x*x+y*y dépasse r*r. Un test supplémentaire stoppe le programme lorsque y dépasse x.

L'intérêt de la méhode tient au fait qu'elle ne met en jeu que des calculs et des tests très rapides. On simplifie encore le programme en remarquant que le carré d'un entier est la somme des impairs de rang au plus égal à cet entier: 1+3=2*2; 1+3+5=3*3, 1+3+5+7=4*4, etc ...

1) VERSION BASIC

La version BASIC, à la fois très rapide et très courte, a été essayée avec succès sur AMSTRAD pour représenter un tore non régulier avec trou, qui comporte la construction de 72 cercles. On a conservé la méthode trigonométrique pour construire l'ellipse lieu des centres, car sa contribution à la durée d'exécution du programme n'est pas critique.

```
10 DEFINT a-z: DEFREAL t
20 CLS:DEG
30 FOR i=0 TO 355 STEP 6
40 x0=270+240*COS(i): y0=200+100*SIN(i): r=(x0+y0)/9
50 GOSUB 80
60 NEXT
70 GOTO 70
```

AMSTRAD

→

```
80 REM cercle de centre x0,y0 et de rayon r
90 x=r: y=0: p=r
100 PLOT x0+x,y0+y: PLOT x0+x,y0-y: PLOT x0-x,y0+y: PLOT x0-x,y0-y
110 PLOT x0+y,y0+x: PLOT x0+y,y0-x: PLOT x0-y,y0+x: PLOT x0-y,y0-x
120 IF y>x THEN RETURN
130 y=y+1:p=p+1-y-y
140 IF p>0 THEN 100
150 x=x-1:p=p+x+x:GOTO 100
```

Temps total d'exécution de ce programme : 75 secondes.

2) VERSION LPB

Pour accélérer encore ce programme, il est tentant de commencer par convertir en langage-machine le sous-programme de construction d'un cercle (lignes 80 à 150 du programme BASIC).

Voici la version en langage LPB :

```
90 DEFW R,X0,Y0,X,Y,X1,X2,Y1,Y2
100 ==========================================
101 REM cercle de centre X0,Y0) et de rayon r
102 ==========================================
100 X=HL=R:Y=HL=0:GOSUB &B906 :REM UPPER ROM ENABLE
110 HL=R: REM P=R
120 PUSH HL:GOSUB 200:POP BC
130 HL=X:DE=Y:A=A OR A:HL=HL SBC DE: IF < THEN &B909 :REM UPPER ROM DISABLE
140 DE+DE+1:HL=HL+1:DE:=:HL:Y=HL:HL=HL+HL:DE:=:HL
150 H=B:C=L:HL=HL SBC DE:IF >=THEN 120 :REM LOOP WHEN P>=0
150 B=H:L=C:X=HL=X-1:HL=HL+HL+BC:GOTO 120 : REM  P=P+2*X
200 REM =============
201 REM PLOT 8 POINTS
202 REM =============
210 DE=X:X1=HL=X0+DE:X2=HL=X0 SBC DE
220 DE=Y:Y1=HL=Y0+DE:Y2=HL=Y0 SBC DE
230 GOSUB 300
240 DE=Y:X1=HL=X0+DE:X2=HL=X0 SBC DE
250 DE=X:Y1=HL=Y0+DE:Y2=HL=Y0 SBC DE
260 GOSUB 300
270 RETURN
300 REM =========================
301 REM PLOT 4 POINTS EN RETANGLE
302 REM =========================
310 DE=X1:HL=Y1:GOSUB &1816
320 DE=X1:HL=Y2:GOSUB &1816
330 DE=X2:HL=Y1:GOSUB &1816
340 DE=X2:HL=Y2:GOSUB &1816
350 RETURN
```

Avec cette routine en LPB, le temps d'exécution du programme complet descend à 13 secondes ■

```
ty kermit.c
/*
 *  K e r m i t  File Transfer Utility
 *
 *  UNIX Kermit, Columbia University, 1981, 1982, 1983
 *      Bill Catchings, Bob Cattani, Chris Maio, Frank da Cruz, Alan Crosswell
 *
 *  Also:   Jim Guyton, Rand Corporation
 *          Walter Underwood, Ford Aerospace
 *
 *  usage:  kermit c [lbe line baud escapechar]        to connect
 *          kermit s [d..iflb line baud] file ...      to send files
 *          kermit r [d..iflb line baud]               to receive files
 *
 *  where   c=connect, s=send, r=receive,
 *          d=debug, i=image mode, f=no filename conversion, l=tty line,
 *          b=baud rate, e=escape char.
 *
 *  For remote Kermit, format is either:
 *          kermit r                                   to receive files
 *  or      kermit s file ...                          to send files
 *
 */
```

KERMIT

```
/*
 *  Modification History:
 *
 *  May 21 84 - Roy Smith (CUCS), strip parity from checksum in rpack()
 *
 *  Oct. 17 Included fixes from Alan Crosswell (CUCCA) for IBM_UTS:
 *          - Changed MYEOL character from \n to \r.
 *          - Change char to int in bufill so getc would return -1 on
 *            EOF instead of 255 (-1 truncated to 8 bits)
 *          - Added read() in rpack to eat the EOL character
 *          - Added fflush() call in printmsg to force the output
 *          NOTE: The last three changes are not conditionally compiled
 *                since they should work equally well on any system.
 *
 *          Changed Berkeley 4.x conditional compilation flag from
 *              UNIX4X to UCB4X.
 *          Added support for error packets and cleaned up the printing
 *              routines.
 */

#include <stdio.h>              /* Standard UNIX definitions */

/* Conditional compilation for different machines/operating systems */
/* One and only one of the following lines should be 1 */

#define UCB4X       1           /* Berkeley 4.x UNIX */
#define TOPS_20     0           /* TOPS-20 */
#define IBM_UTS     0           /* Amdahl UTS on IBM systems */
#define VAX_VMS     0           /* VAX/VMS (not yet implemented) */

/* Conditional compilation for the different Unix variants */
/* 0 means don't compile it, nonzero means do */

#if UCB4X
#define V6_LIBS     0           /* Dont't use retrofit libraries */
#define NO_FIONREAD 0           /* We have ioctl(FIONREAD,...) for flushinput() */
#define NO_TANDEM   0           /* We have TANDEM line discipline (xon/xoff) */
#endif

#if IBM_UTS
#define V6_LIBS     0           /* Don't use retrofit libraries */
#define NO_FIONREAD 1           /* No ioctl(FIONREAD,...) for flushinput() */
#define NO_TANDEM   1           /* No TANDEM line discipline (xon/xoff) */
#endif

#if VAX_VMS
#define V6_LIBS     0           /* Don't use retrofit libraries */
#define NO_FIONREAD 1           /* No ioctl(FIONREAD,...) for flushinput() */
#define NO_TANDEM   1           /* No TANDEM line discipline (xon/xoff) */
#endif

#if V6_LIBS
#include <retrofit/sgtty.h>
#include <retrofit/signal.h>
#include <retrofit/setjmp.h>
#else
#include <signal.h>
#include <setjmp.h>
#endif

#if !(V6_LIBS | VAX_VMS)
#include <sgtty.h>
#endif
```

```c
#if NO_TANDEM
#define TANDEM       0          /* define it to be nothing if it's unsupported */
#endif


/* Symbol Definitions */

#define MAXPACKSIZ  94          /* Maximum packet size */
#define SOH         1           /* Start of header */
#define CR          13          /* ASCII Carriage Return */
#define SP          32          /* ASCII space */
#define DEL         127         /* Delete (rubout) */
#define ESCCHR      '^'         /* Default escape character for CONNECT */

#define MAXTRY      10          /* Times to retry a packet */
#define MYQUOTE     '#'         /* Quote character I will use */
#define MYPAD       0           /* Number of padding characters I will need */
#define MYPCHAR     0           /* Padding character I need (NULL) */

#if IBM_UTS
#define MYEOL       '\r'        /* End-Of-Line character for UTS systems */
#else
#define MYEOL       '\n'        /* End-Of-Line character I need */
#endif

#define MYTIME      10          /* Seconds after which I should be timed out */
#define MAXTIM      60          /* Maximum timeout interval */
#define MINTIM      2           /* Minumum timeout interval */

#define TRUE        -1          /* Boolean constants */
#define FALSE       0


/* Macro Definitions */

/*
 * tochar: converts a control character to a printable one by adding a space.
 *
 * unchar: undoes tochar.
 *
 * ctl:    converts between control characters and printable characters by
 *         toggling the control bit (ie. ^A becomes A and A becomes ^A).
 */
#define tochar(ch)  ((ch) + ' ')
#define unchar(ch)  ((ch) - ' ')
#define ctl(ch)     ((ch) ^ 64 )


/* Global Variables */

int     size,                   /* Size of present data */
        rpsiz,                  /* Maximum receive packet size */
        spsiz,                  /* Maximum send packet size */
        pad,                    /* How much padding to send */
        timint,                 /* Timeout for foreign host on sends */
        n,                      /* Packet number */
        numtry,                 /* Times this packet retried */
        oldtry,                 /* Times previous packet retried */
        ttyfd,                  /* File descriptor of tty for I/O, 0 if remote */
        remote,                 /* -1 means we're a remote kermit */
        image,                  /* -1 means 8-bit mode */
        debug,                  /* indicates level of debugging output (0=none) */
        filnamcnv,              /* -1 means do file name case conversions */
        filecount;              /* Number of files left to send */

char    state,                  /* Present state of the automaton */
        padchar,                /* Padding character to send */
        eol,                    /* End-Of-Line character to send */
        escchr,                 /* Connect command escape character */
        quote,                  /* Quote character in incoming data */
        **filelist,             /* List of files to be sent */
        *filnam,                /* Current file name */
        recpkt[MAXPACKSIZ],     /* Receive packet buffer */
        packet[MAXPACKSIZ];     /* Packet buffer */

FILE    *fp,                    /* File pointer for current disk file */
        *log;                   /* File pointer for Logfile */

jmp_buf env;                    /* Environment ptr for timeout longjump */


/*
 *  m a i n
 *
 *  Main routine - parse command and options, set up the
 *  tty lines, and dispatch to the appropriate routine.
 */
```

```
main(argc,argv)
int argc;                               /* Character pointers to and count of */
char **argv;                               /* command line arguments */
{
    char *ttyname,                         /* tty name for LINE argument */
        *cp;                               /* char pointer */
    int speed,                             /* speed of assigned tty, */
        cflg, rflg, sflg;                  /* flags for CONNECT, RECEIVE, SEND */

    struct sgttyb
        rawmode,                           /* Controlling tty raw mode */
        cookedmode,                        /* Controlling tty cooked mode */
        ttymode;                           /* mode of tty line in LINE option */

    if (argc < 2) usage();                 /* Make sure there's a command line */

    cp = *++argv; argv++; argc -= 2;       /* Set up pointers to args */

/*  Initialize these values and hope the first packet will get across OK */

    eol = CR;                              /* EOL for outgoing packets */
    quote = '#';                           /* Standard control-quote char "#" */
    pad = 0;                               /* No padding */
    padchar = NULL;                        /* Use null if any padding wanted */

    speed = cflg = sflg = rflg = 0;        /* Turn off all parse flags */
    ttyname = 0;                           /* Default is remote mode */

#if UCB4X                                  /* Default to 7-bit masking, CRLF */
    image = FALSE;                         /* translation and filename case */
    filnamcnv = TRUE;                      /* conversion for UNIX systems */
#else
    image = TRUE;                          /* Default to no processing for */
    filnamcnv = FALSE;                     /* non-UNIX systems */
#endif

    escchr = ESCCHR;                       /* Default escape character */

    while ((*cp) != NULL)                  /* Parse characters in first arg. */
        switch (*cp++)
        {
            case 'c': cflg++; break;       /* C = Connect command */
            case 's': sflg++; break;       /* S = Send command */
            case 'r': rflg++; break;       /* R = Receive command */

            case 'd':                      /* D = Increment debug mode count */
                debug++; break;

            case 'f':
                filnamcnv = FALSE;         /* F = don't do case conversion */
                break;                     /*     on filenames */

            case 'i':                      /* I = Image (8-bit) mode */
                image = TRUE; break;       /* (this is default for non-UNIX) */

            case 'l':                      /* L = specify tty line to use */
                if (argc--) ttyname = *argv++;
                else usage();
                if (debug) printf("Line to remote host is %s\n",ttyname);
                break;

            case 'e':                      /* E = specify escape char */
                if (argc--) escchr = **argv++;
                else usage();
                if (debug) printf("Escape char is \"%c\"\n",escchr);
                break;

            case 'b':                      /* B = specify baud rate */
#if UCB4X
                if (argc--) speed = atoi(*argv++);
                else usage();
                if (debug) printf("Line speed to remote host is %d\n",speed);
                break;
#else
                printmsg("Speed setting implemented for Unix only.");
                exit(1);
#endif
        }

/* Done parsing */

    if ((cflg+sflg+rflg) != 1)             /* Only one command allowed */
        usage();
```

```
    if (ttyname)                        /* If LINE was specified, we */
    {                                   /* operate in local mode */
        ttyfd = open(ttyname,2);        /* Open the tty line */
        if (ttyfd < 0)
        {
            printmsg("Cannot open %s",ttyname);
            exit(1);
        }
        remote = FALSE;                 /* Indicate we're in local mode */
    }
    else                                /* No LINE specified so we operate */
    {                                   /* in remote mode (ie. controlling */
        ttyfd = 0;                      /* tty is the communications line) */
        remote = TRUE;
    }

/* Put the proper tty into the correct mode */

    if (remote)                         /* If remote, use controlling tty */
    {
        gtty(0,&cookedmode);            /* Save current mode so we can */
        gtty(0,&rawmode);               /* restore it later */
        rawmode.sg_flags |= (RAW|TANDEM);
        rawmode.sg_flags &= ~(ECHO|CRMOD);
        stty(0,&rawmode);               /* Put tty in raw mode */
    }
    else                                /* Local, use assigned line */
    {
        gtty(ttyfd,&ttymode);
        ttymode.sg_flags |= (RAW|TANDEM);
        ttymode.sg_flags &= ~(ECHO|CRMOD);

#if UCB4X                               /* Speed changing for UNIX only */
        if (speed)                      /* User specified a speed? */
        {
            switch(speed)               /* Get internal system code */
            {
                case 110: speed = B110; break;
                case 150: speed = B150; break;
                case 300: speed = B300; break;
                case 1200: speed = B1200; break;
                case 2400: speed = B2400; break;
                case 4800: speed = B4800; break;
                case 9600: speed = B9600; break;

                default:
                    printmsg("Bad line speed.");
                    exit(1);
            }
            ttymode.sg_ispeed = speed;
            ttymode.sg_ospeed = speed;
        }
#endif /* UCB4X */

        stty(ttyfd,&ttymode);           /* Put asg'd tty in raw mode */
    }


    if (debug)
    {
        printf("Debugging level = %d\n\n",debug);

        if (cflg) printf("Connect command\n\n");
        if (sflg) printf("Send command\n\n");
        if (rflg) printf("Receive command\n\n");
    }

    if (cflg) connect();                /* Connect command */

    if (sflg)                           /* Send command */
    {
        if (argc--) filnam = *argv++;   /* Get file to send */
        else
        {   if (remote)                 /* Restore controlling tty's modes */
                stty(0,&cookedmode);    /* and give error */
            usage();
        }
        fp = NULL;                      /* Indicate no file open yet */
        filelist = argv;                /* Set up the rest of the file list */
        filecount = argc;               /* Number of files left to send */
        if (sendsw() == FALSE)          /* Send the file(s) */
            printmsg("Send failed.");   /* Report failure */
        else                            /*  or */
            printmsg("done.");          /* success */
    }
```

```
    if (rflg)                               /* Receive command */
    {
        if (recsw() == FALSE)               /* Receive the file(s) */
            printmsg("Receive failed.");
        else                                /* Report failure */
            printmsg("done.");              /* or success */
    }

    if (remote) stty(0,&cookedmode);        /* Restore controlling tty's modes */
}


/*
 *  s e n d s w
 *
 *  Sendsw is the state table switcher for sending files.  It loops until
 *  either it finishes, or an error is encountered.  The routines called
 *  by sendsw are responsible for changing the state.
 *
 */

sendsw()
{
    char sinit(), sfile(), sdata(), seof(), sbreak();

    state = 'S';                            /* Send initiate is the start state */
    n = 0;                                  /* Initialize message number */
    numtry = 0;                             /* Say no tries yet */
    while(TRUE)                             /* Do this as long as necessary */
    {
        if (debug) printf("sendsw state: %c\n",state);
        switch(state)
        {
            case 'S':   state = sinit();  break; /* Send-Init */
            case 'F':   state = sfile();  break; /* Send-File */
            case 'D':   state = sdata();  break; /* Send-Data */
            case 'Z':   state = seof();   break; /* Send-End-of-File */
            case 'B':   state = sbreak(); break; /* Send-Break */
            case 'C':   return (TRUE);           /* Complete */
            case 'A':   return (FALSE);          /* "Abort" */
            default:    return (FALSE);          /* Unknown, fail */
        }
    }
}


/*
 *  s i n i t
 *
 *  Send Initiate: send this host's parameters and get other side's back.
 */

char sinit()
{
    int num, len;                           /* Packet number, length */

    if (numtry++ > MAXTRY) return('A');     /* If too many tries, give up */
    spar(packet);                           /* Fill up init info packet */

    flushinput();                           /* Flush pending input */

    spack('S',n,6,packet);                  /* Send an S packet */
    switch(rpack(&len,&num,recpkt))         /* What was the reply? */
    {
        case 'N':  return(state);           /* NAK, try it again */

        case 'Y':                           /* ACK */
            if (n != num)                   /* If wrong ACK, stay in S state */
                return(state);              /* and try again */
            rpar(recpkt);                   /* Get other side's init info */

            if (eol == 0) eol = '\n';       /* Check and set defaults */
            if (quote == 0) quote = '#';

            numtry = 0;                     /* Reset try counter */
            n = (n+1)%64;                   /* Bump packet count */
            return('F');                    /* OK, switch state to F */

        case 'E':                           /* Error packet received */
            prerrpkt(recpkt);               /* Print it out and */
            return('A');                    /* abort */

        case FALSE: return(state);          /* Receive failure, try again */

        default: return('A');               /* Anything else, just "abort" */
    }
}
```

```
/*
 *  s f i l e
 *
 *  Send File Header.
 */

char sfile()
{
    int num, len;                       /* Packet number, length */
    char filnam1[50],                   /* Converted file name */
        *newfilnam,                     /* Pointer to file name to send */
        *cp;                            /* char pointer */

    if (numtry++ > MAXTRY) return('A'); /* If too many tries, give up */

    if (fp == NULL)                     /* If not already open, */
    {   if (debug) printf("   Opening %s for sending.\n",filnam);
        fp = fopen(filnam,"r");         /* open the file to be sent */
        if (fp == NULL)                 /* If bad file pointer, give up */
        {
            error("Cannot open file %s",filnam);
            return('A');
        }
    }

    strcpy(filnam1, filnam);            /* Copy file name */
    newfilnam = cp = filnam1;
    while (*cp != '\0')                 /* Strip off all leading directory */
        if (*cp++ == '/')               /* names (ie. up to the last /). */
            newfilnam = cp;

    if (filnamcnv)                      /* Convert lower case to upper  */
        for (cp = newfilnam; *cp != '\0'; cp++)
            if (*cp >= 'a' && *cp <= 'z')
                *cp ^= 040;

    len = cp - newfilnam;               /* Compute length of new filename */

    printmsg("Sending %s as %s",filnam,newfilnam);

    spack('F',n,len,newfilnam);         /* Send an F packet */
    switch(rpack(&len,&num,recpkt))     /* What was the reply? */
    {
        case 'N':                       /* NAK, just stay in this state, */
            num = (--num<0 ? 63:num);   /* unless it's NAK for next packet */
            if (n != num)               /* which is just like an ACK for */
                return(state);          /* this packet so fall thru to... */

        case 'Y':                       /* ACK */
            if (n != num) return(state); /* If wrong ACK, stay in F state */
            numtry = 0;                 /* Reset try counter */
            n = (n+1)%64;               /* Bump packet count */
            size = bufill(packet);      /* Get first data from file */
            return('D');                /* Switch state to D */

        case 'E':                       /* Error packet received */
            prerrpkt(recpkt);           /* Print it out and */
            return('A');                /* abort */

        case FALSE: return(state);      /* Receive failure, stay in F state */

        default:    return('A');        /* Something else, just "abort" */
    }
}


/*
 *  s d a t a
 *
 *  Send File Data
 */

char sdata()
{
    int num, len;                       /* Packet number, length */

    if (numtry++ > MAXTRY) return('A'); /* If too many tries, give up */
```

```
    spack('D',n,size,packet);               /* Send a D packet */
    switch(rpack(&len,&num,recpkt))         /* What was the reply? */
    {
        case 'N':                           /* NAK, just stay in this state, */
            num = (--num<0 ? 63:num);       /* unless it's NAK for next packet */
            if (n != num)                   /* which is just like an ACK for */
                return(state);              /* this packet so fall thru to... */

        case 'Y':                           /* ACK */
            if (n != num) return(state); /* If wrong ACK, fail */
            numtry = 0;                     /* Reset try counter */
            n = (n+1)%64;                   /* Bump packet count */
            if ((size = bufill(packet)) == EOF) /* Get data from file */
                return('Z');                /* If EOF set state to that */
            return('D');                    /* Got data, stay in state D */

        case 'E':                           /* Error packet received */
            prerrpkt(recpkt);               /* Print it out and */
            return('A');                    /* abort */

        case FALSE: return(state);          /* Receive failure, stay in D */

        default:    return('A');            /* Anything else, "abort" */
    }
}


/*
 *  s e o f
 *
 *  Send End-Of-File.
 */

char seof()
{
    int num, len;                           /* Packet number, length */
    if (numtry++ > MAXTRY) return('A'); /* If too many tries, "abort" */

    spack('Z',n,0,packet);                  /* Send a 'Z' packet */
    switch(rpack(&len,&num,recpkt))         /* What was the reply? */
    {
        case 'N':                           /* NAK, just stay in this state, */
            num = (--num<0 ? 63:num);       /* unless it's NAK for next packet, */
            if (n != num)                   /* which is just like an ACK for */
                return(state);              /* this packet so fall thru to... */

        case 'Y':                           /* ACK */
            if (n != num) return(state); /* If wrong ACK, hold out */
            numtry = 0;                     /* Reset try counter */
            n = (n+1)%64;                   /* and bump packet count */
            if (debug) printf("   Closing input file %s, ",filnam);
            fclose(fp);                     /* Close the input file */
            fp = NULL;                      /* Set flag indicating no file open */

            if (debug) printf("looking for next file...\n");
            if (gnxtfl() == FALSE)          /* No more files go? */
                return('B');                /* if not, break, EOT, all done */
            if (debug) printf("   New file is %s\n",filnam);
            return('F');                    /* More files, switch state to F */

        case 'E':                           /* Error packet received */
            prerrpkt(recpkt);               /* Print it out and */
            return('A');                    /* abort */

        case FALSE: return(state);          /* Receive failure, stay in Z */

        default:    return('A');            /* Something else, "abort" */
    }
}


/*
 *  s b r e a k
 *
 *  Send Break (EOT)
 */

char sbreak()
{
    int num, len;                           /* Packet number, length */
    if (numtry++ > MAXTRY) return('A'); /* If too many tries "abort" */

    spack('B',n,0,packet);                  /* Send a B packet */
    switch (rpack(&len,&num,recpkt))        /* What was the reply? */
    {
        case 'N':                           /* NAK, just stay in this state, */
            num = (--num<0 ? 63:num);       /* unless NAK for previous packet, */
            if (n != num)                   /* which is just like an ACK for */
                return(state);              /* this packet so fall thru to... */
```

```
        case 'Y':                          /* ACK */
            if (n != num) return(state); /* If wrong ACK, fail */
            numtry = 0;                   /* Reset try counter */
            n = (n+1)%64;                 /* and bump packet count */
            return('C');                  /* Switch state to Complete */

        case 'E':                          /* Error packet received */
            prerrpkt(recpkt);             /* Print it out and */
            return('A');                  /* abort */

        case FALSE: return(state);        /* Receive failure, stay in B */

        default:    return ('A');         /* Other, "abort" */
    }
}


/*
 *  r e c s w
 *
 *  This is the state table switcher for receiving files.
 */

recsw()
{
    char rinit(), rfile(), rdata();     /* Use these procedures */

    state = 'R';                         /* Receive-Init is the start state */
    n = 0;                               /* Initialize message number */
    numtry = 0;                          /* Say no tries yet */

    while(TRUE)
    {
        if (debug) printf(" recsw state: %c\n",state);
        switch(state)                    /* Do until done */
        {
            case 'R':   state = rinit(); break; /* Receive-Init */
            case 'F':   state = rfile(); break; /* Receive-File */
            case 'D':   state = rdata(); break; /* Receive-Data */
            case 'C':   return(TRUE);           /* Complete state */
            case 'A':   return(FALSE);          /* "Abort" state */
        }
    }
}


/*
 *  r i n i t
 *
 *  Receive Initialization
 */

char rinit()
{
    int len, num;                        /* Packet length, number */

    if (numtry++ > MAXTRY) return('A'); /* If too many tries, "abort" */

    switch(rpack(&len,&num,packet))      /* Get a packet */
    {
        case 'S':                        /* Send-Init */
            rpar(packet);                /* Get the other side's init data */
            spar(packet);                /* Fill up packet with my init info */
            spack('Y',n,6,packet);       /* ACK with my parameters */
            oldtry = numtry;             /* Save old try count */
            numtry = 0;                  /* Start a new counter */
            n = (n+1)%64;                /* Bump packet number, mod 64 */
            return('F');                 /* Enter File-Receive state */

        case 'E':                        /* Error packet received */
            prerrpkt(recpkt);            /* Print it out and */
            return('A');                 /* abort */

        case FALSE:                      /* Didn't get packet */
            spack('N',n,0,0);            /* Return a NAK */
            return(state);               /* Keep trying */

        default:    return('A');         /* Some other packet type, "abort" */
    }
}
```

```
/*
 *  r f i l e
 *
 *  Receive File Header
 */

char rfile()
{
    int num, len;                       /* Packet number, length */
    char filnam1[50];                   /* Holds the converted file name */

    if (numtry++ > MAXTRY) return('A'); /* "abort" if too many tries */

    switch(rpack(&len,&num,packet))     /* Get a packet */
    {
        case 'S':                       /* Send-Init, maybe our ACK lost */
            if (oldtry++ > MAXTRY) return('A'); /* If too many tries "abort" */
            if (num == ((n==0) ? 63:n-1)) /* Previous packet, mod 64? */
            {                           /* Yes, ACK it again with  */
                spar(packet);           /* our Send-Init parameters */
                spack('Y',num,6,packet);
                numtry = 0;             /* Reset try counter */
                return(state);          /* Stay in this state */
            }
            else return('A');           /* Not previous packet, "abort" */

        case 'Z':                       /* End-Of-File */
            if (oldtry++ > MAXTRY) return('A');
            if (num == ((n==0) ? 63:n-1)) /* Previous packet, mod 64? */
            {                           /* Yes, ACK it again. */
                spack('Y',num,0,0);
                numtry = 0;
                return(state);          /* Stay in this state */
            }
            else return('A');           /* Not previous packet, "abort" */

        case 'F':                       /* File Header (just what we want) */
            if (num != n) return('A');  /* The packet number must be right */
            strcpy(filnam1, packet);    /* Copy the file name */

            if (filnamcnv)              /* Convert upper case to lower */
                for (filnam=filnam1; *filnam != '\0'; filnam++)
                    if (*filnam >= 'A' && *filnam <= 'Z')
                        *filnam |= 040;

            if ((fp=fopen(filnam1,"w"))==NULL) /* Try to open a new file */
            {
                error("Cannot create %s",filnam1); /* Give up if can't */
                return('A');
            }
            else                        /* OK, give message */
                printmsg("Receiving %s as %s",packet,filnam1);

            spack('Y',n,0,0);           /* Acknowledge the file header */
            oldtry = numtry;            /* Reset try counters */
            numtry = 0;                 /* ... */
            n = (n+1)%64;               /* Bump packet number, mod 64 */
            return('D');                /* Switch to Data state */
        case 'B':                       /* Break transmission (EOT) */
            if (num != n) return ('A'); /* Need right packet number here */
            spack('Y',n,0,0);           /* Say OK */
            return('C');                /* Go to complete state */

        case 'E':                       /* Error packet received */
            prerrpkt(recpkt);           /* Print it out and */
            return('A');                /* abort */

        case FALSE:                     /* Didn't get packet */
            spack('N',n,0,0);           /* Return a NAK */
            return(state);              /* Keep trying */

        default:    return ('A');       /* Some other packet, "abort" */
    }
}


/*
 *  r d a t a
 *
 *  Receive Data
 */

char rdata()
{
    int num, len;                       /* Packet number, length */
    if (numtry++ > MAXTRY) return('A'); /* "abort" if too many tries */
```

```
    switch(rpack(&len,&num,packet))     /* Get packet */
    {
        case 'D':                       /* Got Data packet */
            if (num != n)               /* Right packet? */
            {                           /* No */
                if (oldtry++ > MAXTRY)
                    return('A');        /* If too many tries, abort */
                if (num == ((n==0) ? 63:n-1)) /* Else check packet number */
                {                       /* Previous packet again? */
                    spack('Y',num,6,packet); /* Yes, re-ACK it */
                    numtry = 0;         /* Reset try counter */
                    return(state);      /* Don't write out data! */
                }
                else return('A');       /* sorry, wrong number */
            }
            /* Got data with right packet number */
            bufemp(packet,len);         /* Write the data to the file */
            spack('Y',n,0,0);           /* Acknowledge the packet */
            oldtry = numtry;            /* Reset the try counters */
            numtry = 0;                 /* ... */
            n = (n+1)%64;               /* Bump packet number, mod 64 */
            return('D');                /* Remain in data state */

        case 'F':                       /* Got a File Header */
            if (oldtry++ > MAXTRY)
                return('A');            /* If too many tries, "abort" */

            if (num == ((n==0) ? 63:n-1)) /* Else check packet number */
            {                           /* It was the previous one */
                spack('Y',num,0,0);     /* ACK it again */
                numtry = 0;             /* Reset try counter */
                return(state);          /* Stay in Data state */
            }
            else return('A');           /* Not previous packet, "abort" */

        case 'Z':                       /* End-Of-File */
            if (num != n) return('A');  /* Must have right packet number */
            spack('Y',n,0,0);           /* OK, ACK it. */
            fclose(fp);                 /* Close the file */
            n = (n+1)%64;               /* Bump packet number */
            return('F');                /* Go back to Receive File state */

        case 'E':                       /* Error packet received */
            prerrpkt(recpkt);           /* Print it out and */
            return('A');                /* abort */

        case FALSE:                     /* Didn't get packet */
            spack('N',n,0,0);           /* Return a NAK */
            return(state);              /* Keep trying */

        default:     return('A');       /* Some other packet, "abort" */
    }
}

/*
 *  c o n n e c t
 *
 *  Establish a virtual terminal connection with the remote host, over an
 *  assigned tty line.
 */

connect()
{
    int pid,                            /* Holds process id of child */
        connected;                      /* Boolean connect flag */
    char bel = '\07',
        c;

    struct sgttyb
        rawmode,                        /* Controlling tty raw mode */
        cookedmode;                     /* Controlling tty cooked mode */

    if (remote)                         /* Nothing to connect to in remote */
    {                                   /* mode, so just return */
        printmsg("No line specified for connection.");
        return;
    }

    gtty(0,&cookedmode);                /* Save current mode so we can */
    gtty(0,&rawmode);                   /* restore it later */
    rawmode.sg_flags |= (RAW|TANDEM);
    rawmode.sg_flags &= ~(ECHO|CRMOD);
    stty(0,&rawmode);                   /* Put tty in raw mode */
```

```
    pid = fork();            /* Start fork to get typeout from remote host */

    if (pid)                          /* Parent: send type-in to remote host */
    {
        printmsg("connected...\r");
        connected = TRUE;             /* Put us in "connect mode" */
        while (connected)
        {
            read(0,&c,1);             /* Get a character */
            if ((c&0177) == escchr)   /* Check for escape character */
            {
                read(0,&c,1);
                if ((c&0177) == escchr)
                    write(ttyfd,&c,1);
                else
                switch (c&0177)
                {
                    case 'c':
                    case 'C':
                        connected = FALSE;
                        write(0,"\r\n",2);
                        break;

                    case 'h':
                    case 'H':
                        write(0,"\r\nYes, I'm still here...\r\n",26);
                        break;

                    default:
                        write(0,&bel,1);
                        break;
                }
            }
            else
            {                         /* If not escape charater, */
                write(ttyfd,&c,1);    /* write it out */
                c = NULL;             /* Nullify it (why?) */
            }
        }
        kill(pid,9);                  /* Done, kill the child */
        wait(0);                      /* and bury him */
        stty(0,&cookedmode);          /* Restore tty mode */
        printmsg("disconnected.");
        return;                       /* Done */
    }
    else                 /* Child does the reading from the remote host */
    {
        while(1)                      /* Do this forever */
        {
            read(ttyfd,&c,1);
            write(1,&c,1);
        }
    }
}

/*
 *      KERMIT utilities.
 */

clkint()                              /* Timer interrupt handler */
{
    longjmp(env,TRUE);                /* Tell rpack to give up */
}


/*
 *  s p a c k
 *
 *  Send a Packet
 */

spack(type,num,len,data)
char type, *data;
int num, len;
{
    int i;                            /* Character loop counter */
    char chksum, buffer[100];         /* Checksum, packet buffer */
    register char *bufp;              /* Buffer pointer */

    if (debug>1)                      /* Display outgoing packet */
    {
        if (data != NULL)
            data[len] = '\0';         /* Null-terminate data to print it */
        printf("  spack type: %c\n",type);
        printf("         num:  %d\n",num);
        printf("         len:  %d\n",len);
        if (data != NULL)
            printf("        data: \"%s\"\n",data);
    }
```

```
    bufp = buffer;                          /* Set up buffer pointer */
    for (i=1; i<=pad; i++) write(ttyfd,&padchar,1); /* Issue any padding */

    *bufp++ = SOH;                          /* Packet marker, ASCII 1 (SOH) */
    *bufp++ = tochar(len+3);                /* Send the character count */
    chksum  = tochar(len+3);                /* Initialize the checksum */
    *bufp++ = tochar(num);                  /* Packet number */
    chksum += tochar(num);                  /* Update checksum */
    *bufp++ = type;                         /* Packet type */
    chksum += type;                         /* Update checksum */

    for (i=0; i<len; i++)                   /* Loop for all data characters */
    {
        *bufp++ = data[i];                  /* Get a character */
        chksum += data[i];                  /* Update checksum */
    }
    chksum = (((chksum&0300) >> 6)+chksum)&077; /* Compute final checksum */
    *bufp++ = tochar(chksum);               /* Put it in the packet */
    *bufp = eol;                            /* Extra-packet line terminator */
    write(ttyfd, buffer,bufp-buffer+1);     /* Send the packet */
}

/*
 *  r p a c k
 *
 *  Read a Packet
 */

rpack(len,num,data)
int *len, *num;                             /* Packet length, number */
char *data;                                 /* Packet data */
{
    int i, done;                            /* Data character number, loop exit */
    char t,                                 /* Current input character */
        type,                               /* Packet type */
        cchksum,                            /* Our (computed) checksum */
        rchksum;                            /* Checksum received from other host */

#if UCB4X                                   /* TOPS-20 can't handle timeouts... */
    if (setjmp(env)) return FALSE;          /* Timed out, fail */
    signal(SIGALRM,clkint);                 /* Setup the timeout */
    if ((timint > MAXTIM) || (timint < MINTIM)) timint = MYTIME;
    alarm(timint);
#endif /* UCB4X */

    while (t != SOH)                        /* Wait for packet header */
    {
        read(ttyfd,&t,1);
        t &= 0177;                          /* Handle parity */
    }

    done = FALSE;                           /* Got SOH, init loop */
    while (!done)                           /* Loop to get a packet */
    {
        read(ttyfd,&t,1);                   /* Get character */
        if (!image) t &= 0177;              /* Handle parity */
        if (t == SOH) continue;             /* Resynchronize if SOH */
        cchksum = t;                        /* Start the checksum */
        *len = unchar(t)-3;                 /* Character count */

        read(ttyfd,&t,1);                   /* Get character */
        if (!image) t &= 0177;              /* Handle parity */
        if (t == SOH) continue;             /* Resynchronize if SOH */
        cchksum = cchksum + t;              /* Update checksum */
        *num = unchar(t);                   /* Packet number */

        read(ttyfd,&t,1);                   /* Get character */
        if (!image) t &= 0177;              /* Handle parity */
        if (t == SOH) continue;             /* Resynchronize if SOH */
        cchksum = cchksum + t;              /* Update checksum */
        type = t;                           /* Packet type */

        for (i=0; i<*len; i++)              /* The data itself, if any */
        {                                   /* Loop for character count */
            read(ttyfd,&t,1);               /* Get character */
            if (!image) t &= 0177;          /* Handle parity */
            if (t == SOH) continue;         /* Resynch if SOH */
            cchksum = cchksum + t;          /* Update checksum */
            data[i] = t;                    /* Put it in the data buffer */
        }
        data[*len] = 0;                     /* Mark the end of the data */

        read(ttyfd,&t,1);                   /* Get last character (checksum) */
        if (!image) t &= 0177;              /* Handle parity */
        rchksum = unchar(t);                /* Convert to numeric */
        read(ttyfd,&t,1);                   /* get EOL character and toss it */
        if (!image) t &= 0177;              /* Handle parity */
        if (t == SOH) continue;             /* Resynchronize if SOH */
        done = TRUE;                        /* Got checksum, done */
    }
```

```
#if UCB4X
    alarm(0);                       /* Disable the timer interrupt */
#endif

    if (debug>1)                    /* Display incoming packet */
    {
        if (data != NULL)
            data[*len] = '\0';      /* Null-terminate data to print it */
        printf("  rpack type: %c\n",type);
        printf("         num:  %d\n",*num);
        printf("         len:  %d\n",*len);
        if (data != NULL)
            printf("        data: \"%s\"\n",data);
    }
                                    /* Fold in bits 7,8 to compute */
    cchksum = (((cchksum&0300) >> 6)+cchksum)&077; /* final checksum */

    if (cchksum != rchksum) return(FALSE);

    return(type);                   /* All OK, return packet type */
}


/*
 *  b u f i l l
 *
 *  Get a bufferful of data from the file that's being sent.
 *  Only control-quoting is done; 8-bit & repeat count prefixes are
 *  not handled.
 */

bufill(buffer)
char buffer[];                      /* Buffer */
{
    int i,                          /* Loop index */
        t;                          /* Char read from file */
    char t7;                        /* 7-bit version of above */

    i = 0;                          /* Init data buffer pointer */
    while((t = getc(fp)) != EOF)    /* Get the next character */
    {
        t7 = t & 0177;              /* Get low order 7 bits */

        if (t7 < SP || t7==DEL || t7==quote) /* Does this char require */
        {                                    /* special handling? */
            if (t=='\n' && !image)
            {                       /* Do LF->CRLF mapping if !image */
                buffer[i++] = quote;
                buffer[i++] = ctl('\r');
            }
            buffer[i++] = quote;    /* Quote the character */
            if (t7 != quote)
            {
                t = ctl(t);         /* and uncontrolify */
                t7 = ctl(t7);
            }
        }
        if (image)
            buffer[i++] = t;        /* Deposit the character itself */
        else
            buffer[i++] = t7;

        if (i >= spsiz-8) return(i);    /* Check length */
    }
    if (i==0) return(EOF);          /* Wind up here only on EOF */
    return(i);                      /* Handle partial buffer */
}


/*
 *      b u f e m p
 *
 *  Put data from an incoming packet into a file.
 */

bufemp(buffer,len)
char  buffer[];                     /* Buffer */
int   len;                          /* Length */
{
    int i;                          /* Counter */
    char t;                         /* Character holder */

    for (i=0; i<len; i++)           /* Loop thru the data field */
    {
```

```
        t = buffer[i];                    /* Get character */
        if (t == MYQUOTE)                 /* Control quote? */
        {                                 /* Yes */
            t = buffer[++i];              /* Get the quoted character */
            if ((t & 0177) != MYQUOTE)    /* Low order bits match quote char? */
                t = ctl(t);               /* No, uncontrollify it */
        }
        if (t==CR && !image)              /* Don't pass CR if in image mode */
            continue;

        putc(t,fp);
    }
}


/*
 *  g n x t f l
 *
 *  Get next file in a file group
 */

gnxtfl()
{
    if (debug) printf("   gnxtfl: filelist = \"%s\"\n",*filelist);
    filnam = *(filelist++);
    if (filecount-- == 0) return FALSE; /* If no more, fail */
    else return TRUE;                   /* else succeed */
}


/*
 *  s p a r
 *
 *  Fill the data array with my send-init parameters  */

spar(data)
char data[];
{
    data[0] = tochar(MAXPACKSIZ);          /* Biggest packet I can receive */
    data[1] = tochar(MYTIME);           /* When I want to be timed out */
    data[2] = tochar(MYPAD);            /* How much padding I need */
    data[3] = ctl(MYPCHAR);             /* Padding character I want */
    data[4] = tochar(MYEOL);            /* End-Of-Line character I want */
    data[5] = MYQUOTE;                  /* Control-Quote character I send */
}


/*  r p a r
 *
 *  Get the other host's send-init parameters
 *
 */

rpar(data)
char data[];
{
    spsiz = unchar(data[0]);            /* Maximum send packet size */
    timint = unchar(data[1]);           /* When I should time out */
    pad = unchar(data[2]);              /* Number of pads to send */
    padchar = ctl(data[3]);             /* Padding character to send */
    eol = unchar(data[4]);              /* EOL character I must send */
    quote = data[5];                    /* Incoming data quote character */
}


/*
 *  f l u s h i n p u t
 *
 *  Dump all pending input to clear stacked up NACK's.
 *  (Implemented only for Berkeley Unix at this time).
 */

#if UCB4X&(~NO_FIONREAD)
flushinput()
{
    long int count;                     /* Number of bytes ready to read */
    long int i;                         /* Number of bytes to read in loop */

    ioctl(ttyfd, FIONREAD, &count);     /* See how many bytes pending read */
    if (!count) return;                 /* If zero, then no input to flush */

    while (count)                       /* Loop till all are flushed */
    {
        i = (count<sizeof(recpkt)) ?    /* Read min of count and size of */
            count : sizeof(recpkt);     /*  the read buffer */
        read(ttyfd, recpkt, i);         /* Read a bunch */
        count -= i;                     /* Subtract from amount to read */
    }
}
#else
flushinput()             /* Null version for non-Berkeley Unix */
{}
#endif /* UCB4X&(~FIONREAD) */
```

```
/*
 *  Kermit printing routines:
 *
 *  usage  - print command line options showing proper syntax
 *  printmsg -  like printf with "Kermit: " prepended
 *  error - like printmsg if local kermit; sends a error packet if remote
 *  prerrpkt - print contents of error packet received from remote host
 */


/*
 *  u s a g e
 *
 *  Print summary of usage info and quit
 */

usage()
{
#if UCB4X
    printf("Usage: kermit c[lbe line baud esc.char]        (connect mode)\n");
    printf("or:    kermit s[diflb line baud] file ...      (send mode)\n");
    printf("or:    kermit r[diflb line baud]               (receive mode)\n");
#else
    printf("Usage: kermit c[le line esc.char]              (connect mode)\n");
    printf("or:    kermit s[difl line] file ...            (send mode)\n");
    printf("or:    kermit r[difl line]                     (receive mode)\n");
#endif
    exit(1);
}

/*
 *  p r i n t m s g
 *
 *  Print message on standard output if not remote.
 */

/*VARARGS1*/
printmsg(fmt, a1, a2, a3, a4, a5)
char *fmt;
{
    if (!remote)
    {
        printf("Kermit: ");
        printf(fmt,a1,a2,a3,a4,a5);
        printf("\n");
        fflush(stdout);                 /* force output (UTS needs it) */
    }
}

/*
 *  e r r o r
 *
 *  Print error message.
 *
 *  If local, print error message with printmsg.
 *  If remote, send an error packet with the message.
 */

/*VARARGS1*/
error(fmt, a1, a2, a3, a4, a5)
char *fmt;
{
    char msg[80];
    int len;

    if (remote)
    {
        sprintf(msg,fmt,a1,a2,a3,a4,a5); /* Make it a string */
        len = strlen(msg);
        spack('E',n,len,msg);           /* Send the error packet */
    }
    else
        printmsg(fmt, a1, a2, a3, a4, a5);

    return;
}

/*
 *  p r e r r p k t
 *
 *  Print contents of error packet received from remote host.
 */
prerrpkt(msg)
char *msg;
{
    printf("Kermit aborting with following error from remote host:\n%s\n",msg);
    return;
}
```